January 2015

Real-time agility

Core principles and practices for embedded-software teams

Contents

Executive summary

Agile methods are a cohesive set of concepts, principles and practices that form a key part of a continuous engineering capability for product development. These methods address what most consider the bane of software development: poor and changing requirements, short development cycles, long working hours and the steady accretion of system complexity. To solve these challenges, the agile methodology integrates best practices for design and development, resulting in a development process that can reduce costs and improve the quality and predictability of development projects. Agile principles have been applied successfully in many different real-time and embedded markets, such as communications, medical, industrial automation, defense and aerospace. This paper introduces practical core principles and practices, and demonstrates how such approaches can improve project results for embedded-systems development.

Core principles and practices

The IBM Systems Solution Library contains a set of embedded agile practices optimized for the development of software-intensive real-time and embedded systems. The library is founded on the IBM® Harmony/Agile for Embedded Software Development process (hereafter referred to as the IBM Harmony/Agile process), which defines a set of core development principles. Understanding these principles will help you to understand how the workflows are organized, and the underlying reasons for the structure.

IBM Harmony/Agile process core principles

- Your primary goal is to develop working software
- You should always measure progress against the goal, not against the implementation
- Your primary measure of progress is working software
- The best way avoid defects in your software is to avoid putting them there in the first place
- Continuous feedback is crucial
- Five key views define your architecture
- Plan, track and adapt
- The leading cause of project failure is ignoring risk
- Continuous attention to quality is essential
- Modeling is crucial

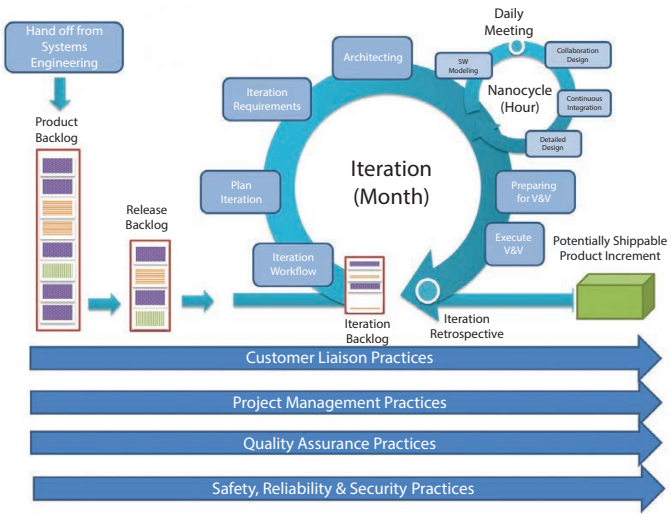

Figure 1. Practice and processes support agile teams

The principles of agile, as defined in the IBM Harmony/Agile process, focus primarily on quality improvement and risk reduction, and only secondarily on team and developer productivity. Improving quality naturally improves productivity through a combination of reduction of rework and decrease in required retesting. The wisdom of this approach is borne out by IBM's success in applying these principles in the field.

Core practices

The core principles of the IBM Harmony/Agile process are enacted by a set of detailed workflows, which guide engineering work in effective ways.

Each flow contains a set of worker tasks, produced or consumed work products, and worker roles and responsibilities. Each practice focuses on a development aspect important for quality management and improvement.

Improperly managed project risk is a common cause of project failure. Risk is, ultimately, the product of two independent factors. The first is the severity of the risk, and its potential impact on the successful completion of a project. For example, the severity of improperly estimating a schedule can be extremely high. Similarly, reliance on technology that proves ineffective in the target system (such as might occur if it is too slow or resource-intensive) can create a high-severity risk. The second factor is how likely a risk is to occur and lead to the failure of a project. If we have a history of producing high-quality software and the team and technology remain relatively unchanged, our exposure to risk is likely to be lower than if we have a history of high defect density, or if we are radically changing the development team, personnel or product technologies.

Risk is all about *stuff you don't actually know* that is crucial for success. One of the key aspects of effective risk management is that risk should be mitigated as early as possible. This maximizes forward progress and minimizes rework. Pounding out millions of lines of code quickly is not beneficial if that code isn't meeting the customer need or if it is full of defects. As you develop code, you need metrics that continually assess its quality during development, rather than in a later testing phase. It is far better to use these practices to avoid putting defects into the code baseline than to spend later effort to track down and repair hundreds or thousands of defects. Most agile practices reduce risk through early, continuous demonstration that the work products in development are of high quality and meet stakeholder needs. Work products that cannot be demonstrated to be of high quality are of little value.

Practice	Risks addressed	Mechanism of risk mitigation
Executable requirements modeling	• Ambiguous requirements • Missing requirements • Conflicting requirements	• Organize requirements into use cases • Build state-based use-case models that demonstrate the interaction of requirements by way of execution
Model-based hand-off from systems engineering	• Loss of fidelity of information during hand-off to software team	• Use principled systems engineering model organization to facilitate hand-off • Hand-off process migrates logical systems engineering work products to physical specifications without loss of fidelity
Dynamic scheduling	• Schedules are inaccurate • Schedules don't reflect changing project needs or conditions • Schedules don't increase in accuracy during the project	• Use BERT scheduling* to get initial best schedule • Create three schedules (Best, Customer and Goal) • Update schedules frequently founded on outcome-based metrics • Use ERNIE estimation† to improve estimation quality • Manage schedules at the project and microcycle levels
High-fidelity modeling	• Design doesn't meet requirements • Design is low quality • Design is not understood by relevant stakeholders • Code does not accurately reflect design • Architectural views are not reflected in the code	• Use UML to model software precisely • Execute and debug models at the model level • Use model-code associativity to synchronize models and code automatically • Generate deployed code from models, when possible
Continuous execution and debugging	• Model and code contain unknown defects	• Execute and debug models at the model level • Use 20- to 60-minute nanocycles for model-code-unit test cycles
Test-driven development	• Model and code contain unknown defects	• Use 20- to 60-minute nanocycles for model-code-unit test cycles • Use model-based testing tooling to create test architectures and execute and analyze test cases
Continuous integration	• Developers' components don't work together properly	• Use daily integrations to ensure different developers' implementations work together • Use shared interface libraries (models) for cross-component interfaces and data types
Incremental development	• System has too many defects • System doesn't meet customer need	• Build the system in 4 - 6 week microcycles • Perform verification and validation at the end of each microcycle • Fix any critical defects before moving to next microcycles • Noncritical defects become work items for subsequent microcycles
Continuous risk mitigation	• Project fails because of unforeseen problems • Potential catastrophic events and conditions are not tracked	• Create risk management plan with appropriate risk data (for example, likelihood, severity, owner, mitigation activity) • Schedule risk mitigation activities • Review risk plan and outcomes weekly and at the end of the microcycle
Incremental process improvement	• Development team fails to mature and improve their productivity and quality • Key roadblocks to project success are not identified and remediated	• Hold increment review at the end of each microcycle and evaluate project (for example, architecture scalability, schedule adherence, quality metrics, productivity metrics) and use this information to dynamically update plans

Practice	Risks addressed	Mechanism of risk mitigation
Event-driven change management	• Needed changes are forgotten or inadequately addressed • Changes are accepted even though they may be high cost and low value	• Put into place change management system • Evaluate change impact before accepting changes • Update relevant documentation (for example, requirements, architecture, test plans, schedule) • Assign and track change until resolution
On-going dependability analysis (safety, reliability and security critical projects only)	• System design or implementation is unsafe, unreliable or insecure	• Perform initial model-based dependability analysis (for example, UML Safety Analysis Profile, UML Security Analysis Profile) • Revisit analysis during architecture and design to ensure dependability goals are met
Apply design patterns for system optimization	• System design isn't coherent • System design isn't understandable • System design is inefficient against relevant design criteria	• Identify optimization criteria • Rank criteria in order of criticality • Identify design solutions (patterns) that optimize the system at acceptable cost • Instantiate design patterns • Test for continued correctness and for optimization goals
Five key views of architecture	• Architecture doesn't meet the system objectives • Important architectural aspects are ignored	• Integrate different views of architecture into the microcycle mission statements • Subsystem architecture • Concurrency architecture • Dependability architecture • Distribution architecture • Deployment architecture • Apply design-pattern practice at the architectural level

Table 1. IBM Harmony/Agile practices for agile embedded-software development

The practices in the IBM Harmony/Agile process[1] primarily focus on risk reduction, although some focus on process improvement. Table 1 shows the most important practices, the risks they address and their mechanisms of action.

When taking an agile approach to embedded software development, there are three time scales of interest. The overall project time frame is called the *macrocycle* and spans the entire length of the project. It is constructed of some startup work (generally referred to as "pre-spiral planning") and a set of microcycles. The middle time scale is the *microcycle*. Each microcycle is 4 - 6 weeks in length and produces a version of the system on which verification and validation is performed. Figure 2 shows the primary activities of the microcycle.

The smallest unit of time is the nanocycle. It is generally between 20 and 60 minutes in length and produces an incremental version of one or more work products that can be assessed for quality. In the high-fidelity modeling nanocycle shown in figure 3, the inner loops result in executable code and unit tests that are created and applied on a highly frequent basis. This work results in models and code that are not only completely in sync, but are also of very high quality and contain few defects.

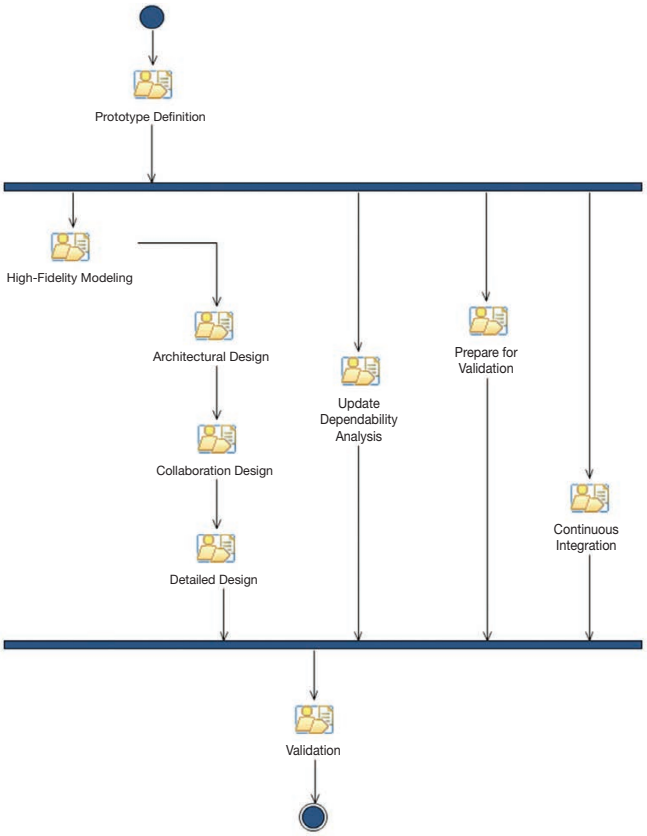

Figure 2. IBM Harmony/Agile microcycle

Agile development solutions from IBM

An efficient agile environment provides assistance in doing engineering work using agile practices, facilitates creation and manipulations of work products, aids in worker collaboration and supports transparent project governance. Such a work environment requires implementation work templates generated from process and practice definition, and tight integration of the tools used by the various roles in the engineering environment.

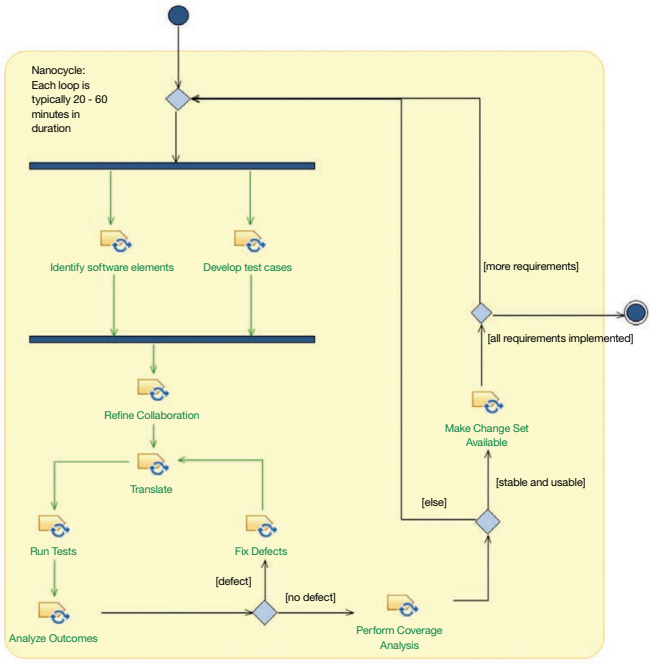

Figure 3. IBM Harmony/Agile nanocycle

The IBM solution for systems and software engineering is integrated on the IBM Jazz™ platform for agile collaborative systems and software delivery. Attuned to the needs of teams of all sizes from small, co-located project teams to large distributed teams, Jazz transforms systems delivery by making it more collaborative, productive and transparent—key aspects of any agile software development environment. You can think of Jazz as an extensible framework that dynamically integrates and synchronizes the people, processes and assets associated with software and systems engineering projects. Unlike the monolithic, closed or point-to-point solutions of the past, Jazz is an open platform that supports the Open Services for Lifecycle Collaboration (OSLC) initiative for improving tool interoperability. Products built on the Jazz platform can leverage a rich set of capabilities for team-based software and systems delivery.

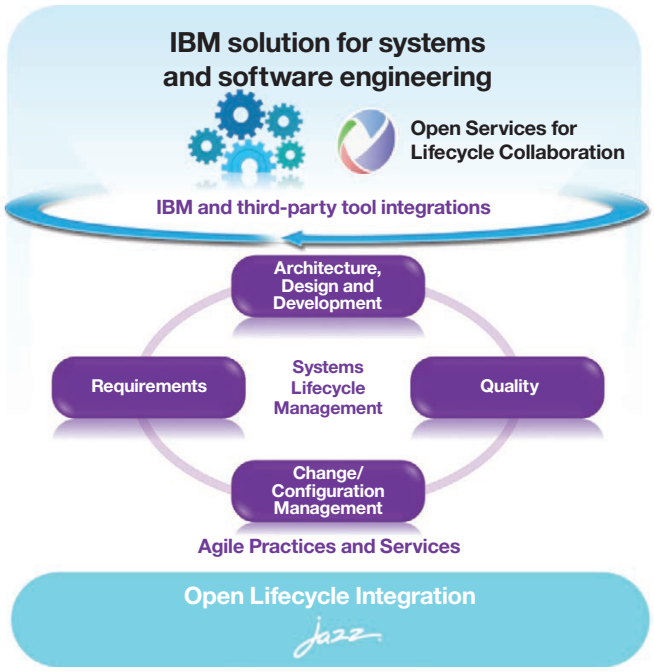

Figure 4. The IBM solution for systems and software engineering

Without integrations across the systems delivery lifecycle, systems teams are left to operate in silos. When silos form, product delivery effectiveness suffers. To respond effectively to ever-changing market needs, systems and software engineering teams must collaborate to manage lifecycle work products. The IBM solution for systems and software engineering provides an integrated system lifecycle management solution to support this collaboration.

Process integration takes place in the IBM solution for systems and software engineering in a couple of different ways. First, the IBM Rational Method Composer tool allows teams to create process and guidance content that is compliant with the Software and Systems Process Engineering Metamodel (SPEM) standard. This includes guidance for workflows, practices, processes, tasks, work products, worker roles and various additions, such as checklists, concept definitions, examples and tool mentors. This content is published in an easily accessible way, usually as a website hosted within the development organization's network.

The second form of process integration is the production of work-task templates from the process definition content, which can be used to support both planning (for example, assign an instance of the "Analyze Use Case" task to Susan) and governance (for example, reporting on the progress of those assigned tasks). This provides managers and developers with a work environment in which tools, work products, processes plans and project governance are tightly integrated. This helps improve efficiency and the visibility of engineering information.

The IBM solution for systems and software engineering provides an integrated solution for the following systems and software engineering team roles.

- *Systems engineers*

The solution provides an integrated and collaborative environment for requirements analysis, architecture management and work, change and configuration management for teams of systems engineers. The leading products are IBM Rational DOORS® or DOORS Next Generation and IBM Rational Rhapsody® with IBM Rational Rhapsody Design Manager for systems engineering tasks, integrated with IBM Rational Team Concert™ for lifecycle management of the work products. IBM Rational Engineering Lifecycle Manager provides visualization, organization and analysis of linked-lifecycle data to enable rapid assessment of the impact of changes in an agile development environment. The integration with IBM Rational Quality Manager enables strong collaboration with systems-validation teams from the start of the project. Work-task templates are created from agile systems engineering practices to support task assignment, execution, and governance.

• *Project, development and test team leads*
Rational Team Concert and Rational Quality Manager provide planning capabilities and work management for systems-delivery teams throughout the project lifecycle and improve transparency through collaboration, automation and reporting on work products and project health.

• *Software engineers*
The IBM solution for systems and software engineering, with Rational Rhapsody integrated with Rational Team Concert in the Eclipse IDE, provides a software development solution for software engineers. This integrates model-driven development, using SysML and UML with the Rational Team Concert capabilities for team collaboration including model configuration management, work items, change sets and continuous software build support. The solution also provides traceability to upstream systems engineering work products in Rational DOORS or DOORS Next Generation and Rational Rhapsody, or downstream traceability for systems integration and validation. Rational Rhapsody Design Manager helps design teams and their stakeholders share, trace, review and manage designs.

• *Software and systems testers*
Rational Quality Manager provides a collaborative environment for test planning, construction and execution, supporting continuous verification within software engineering teams, as well as test management of system validation and acceptance testing. IBM Rational Build Forge® Standard Edition includes test lab automation capabilities that improve the efficiency of systems test labs and manage how test resources are requested and provided.

The interoperability of the IBM solution for systems and software engineering with other tooling means organizations can add agile capabilities while continuing to benefit from existing investments in key development infrastructure, such as the IBM Rational ClearCase® and IBM Rational ClearQuest® or the IBM Rational Synergy and IBM Rational Change configuration and change management environments.

Conclusions

Agile methods have their roots in software development. They are a set of practices and technologies meant first to improve the quality of software work products and secondly to improve the productivity of the engineering teams. Agile methods are based on a set of principles that emphasize work that correlates to quality and de-emphasizes activities that do not. The most important of these practices are incremental development, test-driven development, continuous integration and continuous execution.

Although agile methods come from the IT and enterprise software world, these practices and technologies apply equally well to real-time embedded software development. Through the use of high-fidelity modeling approaches and organizing the workflow to use the agile practices, the quality of software engineering work products can be significantly improved while lowering the engineering cost.

The recommended approach is to adopt agile methods in an agile way. That is, assess where your organization is and where it wants to be, adopt the technology in an incremental fashion using a pilot project with mentoring from experts, monitor success using key performance indicators, reassess, then deploy throughout your organization.

The IBM solution for systems and software engineering provides a combined tooling and agile-practice environment for the various roles within the software-intensive systems engineering environment. Whether starting afresh or using existing tooling investments, tight integration between the core capabilities in the systems engineering environment supports agile teams in their efforts to deliver quality software and systems on time and within budget.

For more information

To learn more about how IBM can help you transition to an agile approach to your real-time, embedded software development, please contact your IBM sales representative or IBM Business Partner, or visit the following IBM solution for systems and software engineering website: **ibm.com**/software/rational/ssesolution/

Additionally, IBM Global Financing can help you acquire the IT solutions that your business needs in the most cost-effective and strategic way possible. We'll partner with credit-qualified clients to customize an IT financing solution to suit your business goals, enable effective cash management, and improve your total cost of ownership. IBM Global Financing is your smartest choice to fund critical IT investments and propel your business forward. For more information, visit: **ibm.com**/financing

About the author

Bruce Powel Douglass, who has a doctorate in neurocybernetics from the USD Medical School, has over 35 years' experience designing safety-critical real-time systems in a variety of hard real-time environments. He has designed and taught courses in agile methods, object-orientation, MDA, real-time systems, and safety-critical systems development, and is the author of over 6000 book pages from a number of technical books including *Real-Time UML*, *Real-Time UML Workshop for Embedded Systems*, *Real-Time Design Patterns*, *Doing Hard Time*, *Real-Time Agility*, *and Design Patterns for Embedded Systems in C*. He is the Chief Evangelist at IBM Rational, where he is a thought leader in the systems space, consulting with and mentoring IBM customers all over the world, represents IBM at many different conferences, and authors tools and processes for the embedded real-time industry. He can be followed on Twitter @IronmanBruce. Papers and presentations are available at his Real-Time UML Yahoo technical group (http://yhoo.it/1EXhJda) and from his IBM page (http://ibm.co/11hr5mR).

Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
January 2015

[1] IBM Harmony/Agile process (http://ibm.co/1ztaqZ3)

* Douglass, Bruce Powel. *Real-Time Agility: The Harmony/ESW Method for Real-Time and Embedded Systems Development*. Addison-Wesley, 2009

[†] Ibid

Please Recycle

RAW14297-USEN-01